Resilient Workflows for Cooperative Design

Toan Nguyen, Laurentiu Trifan, Jean-Antoine Desideri

Resilient Workflows for Cooperative Design

Application of Distributed High-Performance Scientific Computing

Toàn Nguyên, Laurentiu Trifan, Jean-Antoine Désidéri
INRIA
655, avenue de l'Europe
Montbonnot
38334 Saint-Ismier, France
Toan.Nguyen@inrialpes.fr, Laurentiu.Trifan@inrialpes.fr, Jean-Antoine.Desideri@sophia.inria.fr

***Abstract*—This paper describes an approach to extend process modeling for engineering design applications with fault-tolerance and resilience capabilities. It is based on the requirements for application-level error handling, which is a requirement for petascale and exascale scientific computing. This complements the traditional fault-tolerance management features provided by the existing hardware and distributed systems. These are often based on data and operations duplication and migration, and on checkpoint-restart procedures. We show how they can be optimized for high-performance infrastructures. This approach is applied on a prototype tested against industrial testcases for optimization of engineering design artifacts.his electronic document is a "live" template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

Keywords- Workflows; fault-tolerance; resilience; distributed systems; process modeling; high-performance computing; engineering design

I. INTRODUCTION

This paper explores the design, implementation and use of fault-tolerant and resilient simulation platforms. It is based on distributed workflow systems and distributed computing resources [3].

Aiming petascale computing environments, this infrastructure includes heterogeneous distributed hardware and software components. Further, the application codes interact in a timely, secure and effective manner. Additionally, because the coupling of remote hardware and software components are prone to run-time errors, sophisticated mechanisms are necessary to handle unexpected failures at the infrastructure, system and application levels [20][24].

This is also critical for the coupled software that contribute to exascale frameworks [19]. Consequently, specific approaches, methods and software tools are required to handle unexpected errors in large-scale distributed applications.

As mentioned in the Exascale IESP report, current checkpoint/restart and rollback recovery techniques will not fulfill the exascale computing requirements, due in part to their large overhead: "*Because there is no compromise for resilience, the challenges it presents need to be addressed now for solutions to be ready when Exascale systems arrive*" (Section 4.4.1 Resilience, in [19]).

More precisely: "*Resilience is an issue for many efforts. Historically, resilience has not required applications to do anything but checkpoint/restart. Presently, there is a general agreement that the entire software stack, including user and library code, will need to explicitly address resilience beyond the checkpoint/restart approach. We believe this is a uniquely exascale concern and of critical importance.*" (Section 4.5 Summary of X-stack priorities, in [19]).

Among the targets emphasized by the IESP report are (Section 4.4.1 "Resilience", in [19]:

- Fault confinement and local recovery.
- Avoid global coordination towards more local recovery.
- Reducing checkpoint size.
- Language support and paradigm for resilience.
- Dynamic error handling by applications.
- Situational awareness.
- Fault oblivious applications.

This paper addresses three of these issues:

- Promoting *situational awareness* using high-level error handlers defined by the users inside the application workflows.
- Significantly *reducing checkpoint size* used for application recovery, using appropriate heuristics.
- *Dynamic error handling* by executing ad-hoc workflow components that can be dynamically added to the workflow original definitions.

Incidentally, it also addresses the *language support and paradigm* issues, although these benefit directly from the functionalities of the YAWL workflow system that is used [4].

Section II is an overview of related work. Section III is a general description of a sample application, systems and application software. Section IV addresses resilience and exception handling. Section V gives an overview of the implementation, extending the YAWL workflow management system for distributed resilient computing [4]. Section VI is a conclusion.

II. RELATED WORK

Simulation is nowadays a prerequisite for product design and scientific breakthroughs in most application areas, ranging from pharmacy, weather forecast, biology to climate modeling, that all require extensive simulations and testing [6][8]. They often need large-scale experiments, including long-lasting runs in the orders of weeks, tested against petabytes volumes of data and will soon run on exascale supercomputers [10] [11][19].

In such environments, distributed teams usually collaborate on several projects or part of projects. Computerized tools are shared and tightly or loosely coupled [23]. Some codes may be remotely located and non-movable. This requires distributed code and distributed data management facilities. Unfortunately, this is also prone to unexpected errors and breakdowns, e.g., communications, hardware and systems failures.

Data replication and redundant computations have been proposed to prevent from random hardware and communication failures, as well as deadline-dependent scheduling [9].

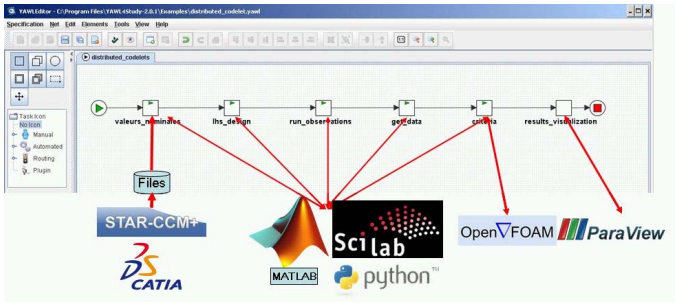

Figure 1. Application testcase.

Hardware and system level fault-tolerance in specific programming environments are also proposed, e.g. Charm++ [5]. Also, middleware and distributed computing systems usually support mechanisms to handle fault-tolerance. They call upon data provenance [12], data replication, redundant code execution, task replication and job migration, e.g., VGrADS [15].

However, erratic application behaviors are seldom addressed [24]. They are due to programming errors, bad application specifications, or poor accuracy and unexpectedly low performance. They also need to be taken into account and handled. This implies evolutions of the application processes in the event of unexpected data values or unexpected control flows. Little has been done so far in this area. The primary concern of the application designers and users has been indeed on efficiency and performance. Therefore, application erratic behavior is usually handled by re-designing and re-programming pieces of code and adjusting parameter values and bounds. This usually requires the simulations to be stopped and restarted [15]. This approach is inadequate when applications run several days and weeks.

Departing from these solutions, a dynamic approach is presented in the following sections (Sections IV and V). It supports the evolution of the application behavior using the introduction of new exception handling rules at run-time by the users, based on the observed (and possibly unexpected) data values. The running workflows do not need to be aborted, as new rules can be added at run-time without stopping the executing workflows [13]. At worst, they need to be paused.

This allows on-the-fly management of unexpected events. It allows also a continuous evolution of the applications, supporting their adaptation to the occurrence of unforeseen events and situations. As new situations arise and new data values appear, new rules can be added to the workflows that will permanently be taken into account in the future. These evolutions are dynamically plugged-in to the workflows, without the need to stop the running applications [13]. The overall application logic is therefore unchanged. This guarantees a continuous adaptation to new situations without the need to redesign the existing workflows, thus promoting *situational awareness*.

Further, because exception-handling codes are themselves defined by new specific workflows plug-ins, the user interface to the applications remains unchanged [14].

Also, checkpoint/restart procedures are addressed by reducing significantly the number of necessary checkpoints, using a new scheme called *asymmetric checkpoints* [3]. This addresses the two critical concerns for *checkpoint size reduction* and the *reduction of restart delays* in large-scale and exascale applications [19].

III. APPLICATION TESTCASE

A. Example

An overview of a running testcase is presented here. It deals with the optimization of a car air-conditioning duct. The goal is to optimize the air flow inside the duct, maximizing the throughput and minimizing the air pressure and air speed discrepancies inside the duct. This example is provided by a car manufacturer and involves industry partners, e.g., software vendors, as well as optimization research teams (Figure 1).

The testcase is a dual faceted 2D and 3D example. Each facet involves different software for CAD modeling, e.g. CATIA and STAR-CCM+, numeric computations, e.g., Matlab, Python and Scilab, flow computations, e.g., OpenFOAM and visualization, e.g., ParaView (Figure 1).

The testcase is deployed using the YAWL workflow management system [4]. The goal is to distribute the testcase on various partners' locations where the different software are running (Figure 2). In order to support this distributed computing approach, an open source middleware is used [17].

B. Application Workflow

In order to provide a simple and easy-to-use interface to the computing software, the YAWL workflow management system is used (Figure 1). It supports high-level graphic specifications for application design, deployment, execution and monitoring. It also supports the modeling of business organizations and interactions among heterogeneous software components.

Indeed, the example testcase described above involves several codes written in Matlab, OpenFOAM and displayed using ParaView. The 3D testcase facet involves CAD files generated using CATIA and STAR-CCM+, flow calculations using OpenFOAM, Python scripts and visualization with ParaView. Future testcases will also require the use of the Scilab toolbox [16].

Because proprietary software are used, as well as open-source and in-house research codes, a secured network of connected computers is made available to the users, based on the ProActive middleware [17].

This network is deployed on the various partners' locations throughout France. Web servers accessed through the ssh protocol are used for the proprietary software running on dedicated servers, e.g., CATIA v5 and STAR-CCM+.

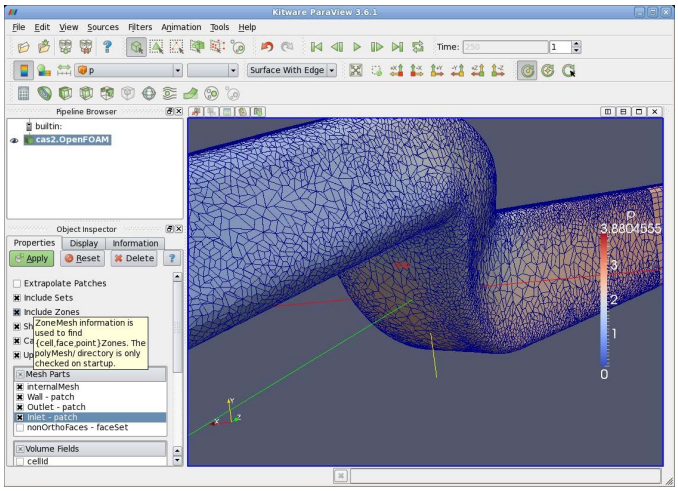

Figure 2. The optimization testcase.

A powerful feature of the YAWL workflow system is that composite workflows can be defined hierarchically [4]. They can invoke external software, i.e., pieces of code written in whatever language is used by the users. They are called by custom YAWL services or local shell scripts. Web Services can also be invoked. Although custom services need Java classes to be implemented, all these features are natively supported in YAWL.

YAWL thus provides an abstraction layer that helps the users design complex applications that may involve a large number of distributed components. Further, the workflow specifications allow alternative execution paths which may be chosen automatically or manually, depending on the data values, as well as parallel branches, conditional branching and loops. Also, multiple instance tasks can execute in parallel for different data values [25]. Combined with the run-time addition of code using the corresponding dynamic selection procedures, as well as new exception handling procedures, a very powerful environment is provided to the users [4]. More details are given in Section V below.

IV. RESILIENCE

A. Rationale

Resilience is commonly defined as "the ability to bounce back from tragedy" and as "resourcefulness" [18]. It is defined here as the ability for the applications to handle correctly unexpected run-time situations, possibly – but not necessarily – with the help of the users.

Usually, hardware, communication and software failures are handled using hard-coded fault-tolerance software [15]. This is the case for communication software and for middleware that take into account possible computer and network breakdowns at run-time. These mechanisms use for example data and packet replication and duplicate code execution to cope with these situations [5].

However, when unexpected situations occur at run-time, which are due to unexpected data values and application erratic behavior, very few options are offered to the users: ignore them or abort the execution, analyze the errors and later modify and restart the applications.

Optimized approaches can be implemented in such cases trying to reduce the amount of computations to be re-run, or anticipating potential discrepancies by multiplying some critical instances of the same computations. This latter approach can rely on statistical estimations of failures. Another approach for anticipation is to prevent total loss of computations by duplicating the calculations that are running on presumably failing nodes [9].

While these approaches deal with hardware and system failures, they do not cope with application failures. These can originate from:

- Incorrect or incomplete specifications.
- Incorrect programming.
- Incorrect anticipation of data behavior, e.g., out-of-bounds data values.
- Incorrect constraint definitions, e.g., approximate boundary conditions.

To cope with this aspect of failures, we introduce an application-level fault management that we call *resilience*. It provides the ability for the applications to survive, i.e., to restart, in spite of their erroneous prevailing state. In such cases, new handling codes can be introduced dynamically by the users in the form of specific new component workflows.

This requires a roll-back to a consistent state that is defined by the users at critical checkpoints.

In order to do this efficiently, a mechanism is implemented to reduce the number of necessary checkpoints. It is based on user-defined rules. Indeed, the application designers and users are the only ones to have the expertise required to define appropriate corrective actions and characterize the critical checkpoints. No automatic mechanisms can be substituted for them, as is the case in hardware and system failures. It is generally not necessary to introduce checkpoints systematically, but only at specific locations of the application processes, e.g., only before parallel branches of the applications. We call these *asymmetric checkpoints* [3].

B. *Exception handling*

The alternative used proposed here to cope with unexpected situations is based on the dynamic selection and exception handling mechanism featured by YAWL [13].

It provides the users with the ability to add at run-time new rules governing the application behavior and new pieces of code that will take care of the new situations.

For example, it allows for the runtime selection of alternative workflows, called worklets, based on the current (and possibly unexpected) data values. The application can therefore evolve over time without being stopped. It can also cope later with the new situations without being altered. This refinement process is therefore lasting over time and the obsolescence of the original workflows reduced.

The new worklets are defined and inserted in the original application workflow using the standard specification approach used by YAWL (Figure 2).

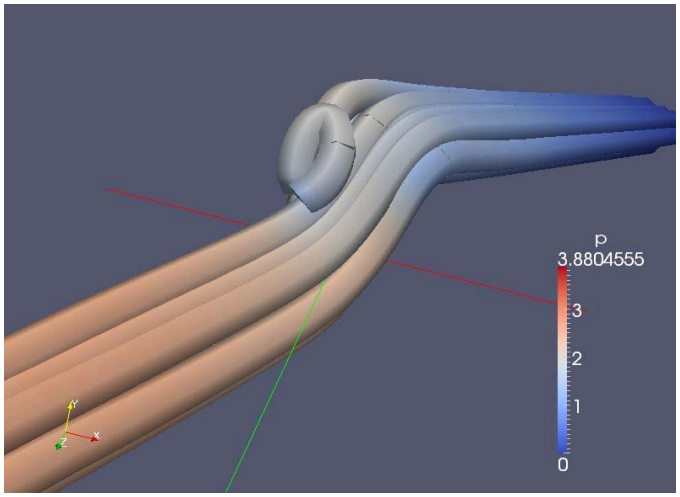

Figure 3. Pressure flow in the air-conditioning duct.

Because it is important that monitoring long-running applications be closely controlled by the users, this dynamic selection and exception handling mechanism also requires a user-defined probing mechanism that provides with the ability to suspend, evolve and restart the code dynamically.

For example, if the output pressure of an air-conditioning pipe is clearly off limits during a simulation run, the user must be able to suspend it as soon as he is aware of that situation. He can then take corrective actions, e.g., suspending the simulation, modifying some parameters or value ranges and restarting the process immediately. These actions can be recorded as new execution rules, stored as additional process description and invoked automatically in the future.

These features are used to implement the applications erratic behavior manager. This one is invoked by the users to restart the applications at the closest checkpoints after corrective actions have been manually performed, if necessary, e.g., modifying boundary conditions for some parameters. Because they have been defined by the users at critical locations in the workflows, the checkpoints can be later chosen automatically among the available asymmetric checkpoints available that are closest to the failure location in the workflow.

V. IMPLEMENTATION

A. *The YAWL workflow system*

Workflows systems are the support for many e-Science applications [1][6][8]. Among the most popular systems are Taverna, Kepler, Pegasus, Bonita and many others [11][15]. They complement scientific software environments like Dakota, Scilab and Matlab in their ability to provide complex application factories that can be shared, reused and evolved. Further, they support the incremental composition of hierarchic composite applications. Providing a control flow approach, they also complement the usual dataflow approach used in programming toolboxes. Another bonus is that they provide seamless user interfaces, masking technicalities of distributed, programming and administrative layers, thus allowing the users and experts to concentrate on their areas of interest.

The OPALE project at INRIA [27] is investigating the use of the YAWL workflow management system for distributed multidiscipline optimization [3]. The goal is to develop a resilient workflow system for large-scale optimization applications. It is based on extensions to the YAWL system to add resilience and remote computing facilities for deployment on high-performance distributed infrastructures. This includes large-PC clusters connected to broadband networks. It also includes interfaces with the Scilab scientific computing toolbox [16] and the ProActive middleware [17].

Provided as an open-source software, YAWL is implemented in Java. It is based on an Apache server using Tomcat and Apache's Derby relational database system for persistence. YAWL is developed by the University of Eindhoven (NL) and the University of Brisbane (Australia). It runs on Linux, Windows and MacOS platforms [25]. It allows complex workflows to be defined and supports high-level constructs (e.g., XOR- and OR-splits and joins, loops, conditional control flow based on application variables values, composite tasks, parallel execution of multiple instances of tasks, etc) through high-level user interfaces (Figure 4).

Formally, it is based on a sound and proven operational semantics extending the *workflow patterns* of the Workflow Management Coalition [21] and implemented by colored Petri nets.

Designed as an open platform, YAWL supports natively interactions with external and existing software and application codes written in any programming languages, through shell scripts invocations, as well as distributed computing through Web Services (Figure 5).

It includes a native Web Services interface, custom services invocations through *codelets*, as well as rules, powerful exception handling facilities, and monitoring of workflow executions [13].

Further, it supports dynamic evolution of the applications by extensions to the existing workflows through *worklets*, i.e.,

on-line inclusion of new workflow components during execution [14].

It supports automatic and step-by-step execution of the workflows, as well as persistence of (possibly partial) executions of the workflows for later resuming, using its internal database system. It also features extensive event logging for later analysis, simulation, configuration and tuning of the application workflows.

Additionally, YAWL supports extensive organizations modeling, allowing complex collaborative projects and teams to be defined with sophisticated privilege management: access rights and granting capabilities to the various projects members (organized as networked teams of roles and capabilities owners) on the project workflows, down to individual components, e.g., edit, launch, pause, restart and abort workitems, as well as processing tools and facilities [25].

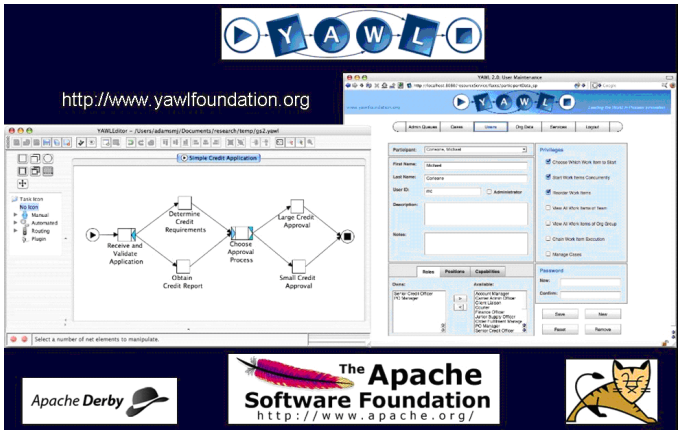

Figure 4. The YAWL interfaces.

Current experiments include industrial testcases, involving the connection of the Matlab, Scilab, Python, ParaView and OpenFOAM software to the YAWL platform [3]. The YAWL workflow system is used to define the optimization processes, include the testcases and control their execution: this includes reading the input data (StarCCM+ files), the automatic invocation of the external software and automatic control passing between the various application components, e.g., Matlab scripts, OpenFOAM, ParaView, ... (Figure 1).

B. *Exception handling*

The exception handlers are automatically tested by the YAWL workflow engine when the corresponding tasks are invoked. This is standard in YAWL and constraint checking can be activated and deactivated by the users [4].

For example, if a particular workflow task WT invokes an external EXEC code through a shell script SH (Figure 6) using a standard YAWL *codelet*, an exception handler EX can be implemented to prevent from undesirable situations, e.g., infinite loops, unresponsive programs, long network delays, etc. Application variables can be tested, allowing for very close monitoring of the applications behavior, e.g., unexpected values, convergence rates for optimization programs, threshold transgressions, etc.

A set of rules (RDR) is defined in a standard YAWL *exlet* attached to the task WT and defines the exception handler EX. It is composed here of a constraint checker CK, which is automatically tested when executing the task WT. A compensation action CP triggered when a constraint is violated and a notifier RE warning the user of the exception. This is used to implement resilience (Section C, below).

The constraint violations are defined by the users and are part of the standard exception handling mechanism provided by YAWL. They can attach sophisticated exception handlers in the form of specific *exlets* that are automatically triggered at runtime when particular user-defined constraints are violated. These constraints are part of the RDR attached to the workflow tasks.

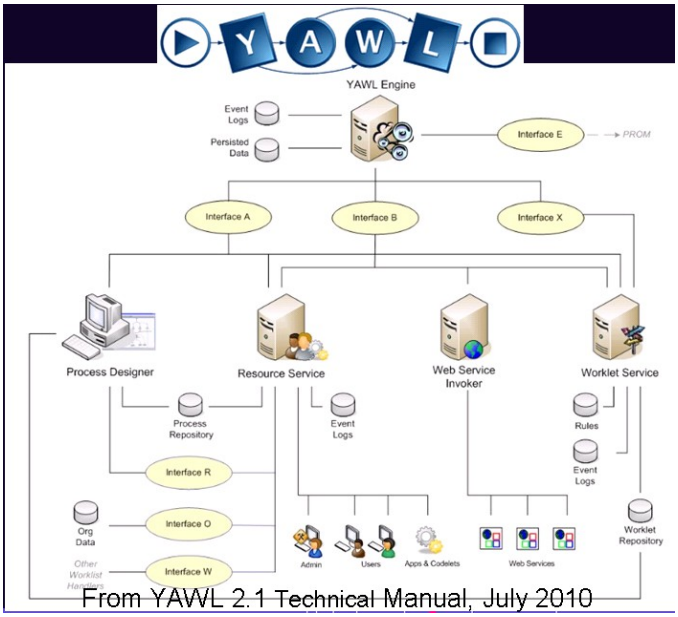

Figure 5. The YAWL archiitecture.

Resilience is the ability for applications to handle unexpected behavior, e.g., erratic computations, abnormal result values, etc. It is inherent to the applications logic and programming. It is therefore different from systems or hardware errors and failures. The usual fault-tolerance mechanisms are therefore inappropriate here. They only cope with late symptoms, at best.

C. *Resilience*

New mechanisms are therefore required to handle logic discrepancies in the applications, most of which are only discovered incrementally during the applications life-time, whatever projected exhaustive details are included at the application design time.

It is therefore important to provide the users with powerful monitoring features and to complement them with dynamic tools to evolve the applications specifications and behavior

according to the future erratic behavior that will be observed during the application life-time.

The exception handlers are used to trigger the resilience mechanism when appropriate.

This is implemented using in the YAWL workflow system the so-called "dynamic selection and exception handling mechanism" [4]. It supports:

- Application update using dynamically added rules specifying new worklets to be executed, based on data values and constraints.
- The persistence of these new rules to allow applications to handle correctly the future occurrences of the new cases.
- The dynamic extension of these sets of rules.
- The definition of the new worklets to be executed, using the native framework provided by the YAWL specification editor: the new worklets are new component workflows attached to the global composite application workflows [13].
- Worklets can invoke external programs written in any programming language through shell scripts, custom service invocations and Web Services [14].

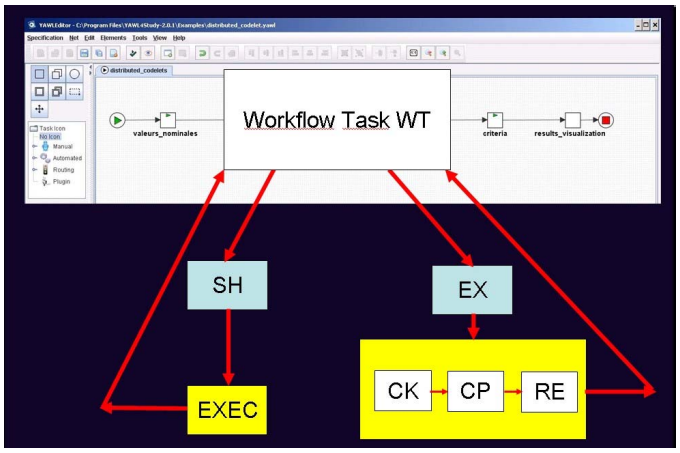

Figure 6. Exception handler associated with a workflow task.

In order to implement resilience, two particular YAWL features are used:

- Ripple-down-rules (RDR) which are handlers for exception management.
- Worklets, which are actions to be taken when exceptions or specific events occur.

The RDR define the decision process which is run to decide which worklet to use in specific circumstances.

VI. CONCLUSION

This paper presents an experiment for deploying a distributed simulation platforms. It uses a network of high-performance computers connected by a middleware layer. Users interact dynamically with the applications using a distributed workflow system. It allows them to define, deploy, evolve and control the application executions.

A significant bonus of this approach is that besides fault-tolerance provided by the middleware, which addresses communication, hardware and system failures, the users can define and handle dynamically, i.e., at run-time, the application failures at the workflow specification level.

This addresses four major concerns that impact exascale application frameworks [19]:

- Reduced checkpoint size.
- Language support and paradigm for resilience.
- Dynamic error handling.
- Situational awareness.

A new abstraction layer is introduced to answer the need for *situational awareness* [19], in order to cope with the application errors at run-time. Indeed, these errors do not necessarily result from programming and design errors. They may also result from unforeseen situations, data values and boundary conditions that could not be envisaged at first. This is often the case for simulations due to the experimental nature of the applications, e.g., discovering the behavior of the system being simulated, like unusual flight dynamics: characterization of the stall behavior of an aircraft for various load and balance profiles at the limits of its flight envelope [2].

To answer the requirement for *reduced checkpoint size* mentioned in [19], the approach presented here also supports resilience using an *asymmetric checkpoints* mechanism described elsewhere [3]. This feature allows for efficient handling mechanisms to restart only those parts of an application that are characterized by the users as critical when treating erratic and unexpected behaviors. It therefore also addresses the *restart delays reduction*.

Further, this approach can evolve dynamically, i.e., when applications are running. This uses the dynamic selection and exception handling mechanism in the YAWL workflow system [4]. Should unexpected situations occur, it allows for new rules and new exception handlers to be plugged-in at run-time by the application designers and the users. This answer the need for *dynamic error handling* at run-time.

Additionally, the requirement for *language support and paradigm for resilience* in [19] is also addressed, using the error handlers plugged into the application workflows in the form of new component workflows. It therefore provides a homogeneous, dynamic and high-level user interface.

ACKNOWLEDGMENT

This work is supported by the French *Agence Nationale de la Recherche* (ANR), contract ANR-08-COSI-007 for the OMD2 project ("*Optimisation Mutlidiscipline Distribuée*").

REFERENCES

[1] Y.Simmhan, R. Barga, C. van Ingen, E. Lazowska and A. Szalay "Building the Trident scientific workflow workbench for data management in the cloud". In proceedings of the 3rd Intl. Conf. on Advanced Engineering Computing and Applications in Science. Sliema (Malta). October 2009. pp 132-138.

[2] A. Abbas, "High computing power: a radical change in circraft design process", In proceedings of the 2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics. Harbin (China) April 2009.

[3] T. Nguyên and J-A Désidéri, "Dynamic resilient workflows for collaborative design", In proceedings of the 6th Intl. Conf. on Cooperative Design, Visualization and Engineering. Luxemburg. September 2009. Springer-Verlag. LNCS 5738, 2009, pp. 341–350.

[4] A.H.M ter Hofstede, W. Van der Aalst, M. Adams and N. Russell, "Modern business process automation: YAWL and its support environment", Springer, 2010.

[5] D. Mogilevsky,G.A. Koenig and Y. Yurcik, "Byzantine anomaly testing in Charm++: providing fault-tolerance and survivability for Charm++ empowered clusters", In proceedings of the 6th IEEE Intl. Symp. On Cluster Computing and Grid Workshops. CCGRIDW'06. Singapore. May 2006. pp146-154.

[6] E. Deelman and Y. Gil., "Managing large-scale scientific workflows in distributed environments: experiences and challenges", In proceedings of the 2nd IEEE Intl. Conf. on e-Science and the Grid. Amsterdam (NL). December 2006. pp 26-32.

[7] Oracle Corp. "Oracle VM VirtualBox user manual", version 3.2.0, May 2010. Also: http://www.virtualbox.org.

[8] M. Ghanem, N. Azam, M. Boniface and J. Ferris, "Grid-enabled workflows for industrial product design", In proceedings of the 2nd Intl. Conf. on e-Science and Grid Computing. Amsterdam (NL). December 2006. pp 88-92.

[9] G. Kandaswamy, A. Mandal and D.A. Reed, "Fault-tolerant and recovery of scientific workflows on computational grids", In proceedings of the 8th Intl. Symp. On Cluster Computing and the Grid. 2008. pp 264-272.

[10] H. Simon. "Future directions in high-performance computing 2009-2018". Lecture given at the ParCFD 2009 Conference. Moffett Field (CA). May 2009.

[11] J. Wang, I. Altintas, C. Berkley, L. Gilbert and M.B. Jones, "A high-level distributed execution framework for scientific workflows", In proceedings of the 4th IEEE Intl. Conf. on eScience. Indianapolis (IN). December 2008. pp 156-164.

[12] D. Crawl and I. Altintas, "A provenance-based fault tolerance mechanism for scientific workflows", In proceedings of the 2nd Intl. Provenance and Annotation Workshop. IPAW 2008. Salt Lake City (UT). June 2008. Springer. LNCS 5272. pp 152-159.

[13] M. Adams, "Facilitating dynamic flexibility and exception handling for workflows", PhD Thesis, Queensland University of Technology, Brisbane (Aus.), 2007.

[14] M. Adams and L. Aldred, "The worklet custom service for YAWL, installation and user manual", Beta-8 Release, Technical Report, Faculty of Information Technology, Queensland University of Technology, Brisbane (Aus.), October 2006.

[15] L. Ramakrishnan et al., "VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance", Proc. ACM SC'09 Conf. Portland (Or.), November 2009.

[16] M. Baudin, "Introduction to Scilab", Consortium Scilab. January 2010. Also: http://wiki.scilab.org/

[17] F. Baude et al., "An efficient framework for running applications on clusters, grids and clouds", in "Cloud Computing: principles, systems and applications", Springer, 2010.

[18] http://edition.cnn.com/2009/TRAVEL/01/20/mumbai.overview/ last accessed: 07/07/2010.

[19] J. Dongarra, P. Beckman et al. "The International Exascale Software Project roadmap". University of Tennessee EECS Technical report UT-CS-10-654. May 2010. Available at: http://www.exascale.org/

[20] R. Gupta, P. Beckman et al. "CIFTS : a Coordinated Infrastructure for Fault-Tolerant Systems". Proc. 38th Intl. Conf. Parallel Processing Systems. Vienna (Au). September 2009.

[21] The Workflow Management Coalition. http://www.wfmc.org

[22] D. Abramson, B. Bethwaite et al. "Embedding optimization in computational science workflows". Journal of Computational Science 1 (2010). pp 41-47. Elsevier.

[23] A.Bachmann, M. Kunde et al. "Advances in generalization and decoupling of software parts in a scientific simulation workflow system". Proc. 4th Intl. Conf. Advanced Engineering Computing and Applications in Sciences. Florence (I). October 2010.

[24] R. Duan, R. Prodan et al. "DEE: a distributed fault tolerant workflow enactment engine for grid computing". Proc. 1st. Intl. Conf. on High-Performance Computing and Communications. Sorrento (I). LNCS 3726. September 2005.

[25] The YAWL Foundation: http://www.yawlfoundation.org/software/

[26] T. Nguyên, L. Trifan, J-A Désidéri. "A distributed workflow platform for simulation". Proc. 4th Intl. Conf on Advanced Engineering Computing and Applications in Sciences. Florence (I). October 2010.

[27] The OPALE project at INRIA: http://www-opale.inrialpes.fr